*Automation and Remote Control, Vol. 64, No. 3, 2003, pp. 493–504. Translated from Avtomatika i Telemekhanika, No. 3, 2003, pp. 163–176.*
*Original Russian Text Copyright © 2003 by Erzin, Cho.*

## COMPUTING TECHNIQUES IN AUTOMATIC CONTROL

# Skew Minimization Problem with Possible Sink Displacement[1]

**A. I. Erzin*** **and J. D. Cho****

**Sobolev Institute of Mathematics, Siberian Branch, Russian Academy of Sciences, Novosibirsk, Russia*
***Sungkyunkwan University, Suwon, South Korea*

Received July 16, 2002

**Abstract**—In applications such as design of integrated circuits (chips), it is sometimes required to connect the terminals (receivers) and the central vertex (source) by a weight-minimal tree where the signal delays between the source and the terminals are the same or differ by a minimal value. Simple necessary conditions for existence of the desired tree, heuristic rules for displacement of the terminals for which these conditions are not satisfied, and a new polynomial algorithm to determine an approximate solution were presented.

## 1. FORMULATION OF THE PROBLEM

As the need for high-speed integrated circuits (chips) is growing from one year to the next, design of their signal systems becomes pivotal. The signal network is responsible for clocking the computer system. The command signal is generated outside the chip and fed into it through the input (source or root). Each functional element for which the signal is destined is connected to the source via a signal network. Each element performs a series of logic operations (functions) and until the beginning of the next cycle of computations waits for a signal to transmit the results to other element(s). Thereby the flows of information within the computer system are checked. The *clock skew* is the maximum difference between the instants of signal arrivals to different system components. Its increase results in lower speed of computations. In the existing systems where the size of elements is substantially smaller than one micron, clock skew is one of the main factors defining operation of the system. It reduces the clock frequency because the time between two successive signals must be increased so as to enable all system components to receive the signal. It is currently believed that the clock skew in the high-speed circuits should not exceed 5% of the maximum (critical) signal transmission time (critical delay).

As will be shown in what follows, it does not always happen that there are clock trees with zero clock skew. In such an event, it is sometimes admitted to displace the terminals from their initial positions at a distance of at most two units. The possibility of displacing the terminals makes the construction of the zero-skew clock tree especially difficult. The problem of constructing a min-skew clock tree is known to be NP-hard [3] even without displacement of the terminals.

The problem of clocking the arrivals of command signals to the elements of integrated circuit can be stated as follows. Each terminal (circuit element) executes certain operations. All terminals execute part of the general program, and their operation must be coordinated. The requirement of simultaneous arrival of signals to all terminals can be satisfied if one constructs a tree where all paths from the source to the terminals have identical delays. Additionally, from all the admissible trees the minimum-weight (cost) one must be selected in order to minimize the space occupied. In

[1] This work was supported in part by the grants of KISTEP'99 and the Russian Foundation for Basic Research, project no. 02-01-00977.

0005-1179/03/6403-0493$25.00 © 2003 MAIK "Nauka/Interperiodica"

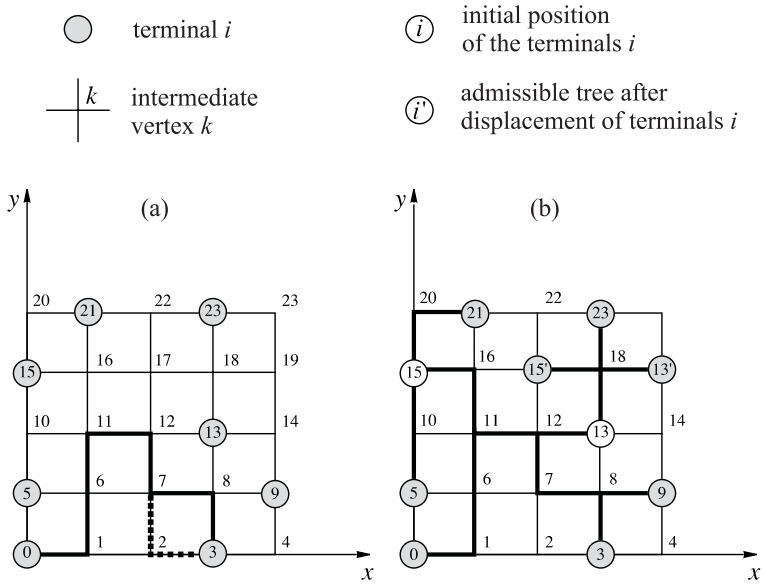


**Fig. 1.** Example of the problem. (a) Initial position of the terminals. (b) Admissible tree after displacement of the terminals.

doing so, the tree edges are routed on a uniform rectangular planar grid (in particular, the degree of each graph vertex does not exceed four). Consequently, the admissible tree does not necessarily exist.

In mathematical terms, without displacement of terminals the problem lies in constructing a minimal rectilinear Steiner tree where the distances (along the tree) from the root to each terminal are equal to the radius of the grid graph. So, let an $n$-vertex ($n = |V|$, $V$ is the vertex set) rectangular grid with unit distance between the neighboring horizontal and vertical lines be given, and for simplicity, let it be located in the first quadrant, the source vertex 0 (root) being at the origin. Let the terminals making up the set $V' \subseteq V$ be located arbitrarily at the nodes of the grid (Fig. 1a). It is required to construct a tree $T$ of the minimum weight which lies completely in the first quadrant and connects all terminals to the root by paths of the same length (with the same number of grid edges) equal to the rectangular distance $L$ to the outermost terminal. The weight of the tree is equal to the number of grid edges included in it. If it is impossible to construct such a tree, then it is admitted to move the terminals at most at two units (edges) away from the initial place. It is suggested to decompose the solution of the problem into two stages. At the first stage, we try to construct the desired tree without moving the terminals. If this turns out to be impossible, then at the second stage some terminals are displaced.

The mathematical model of the first stage (without displacement of terminals) can be set down as follows:

$$|\{(i,j) \in T\}| \to \min_{T}; \tag{1}$$

$$|\{(i,j) \in P_k(T)\}| = L, \quad k \in V', \tag{2}$$

where $P_k(T)$ is the path from the $k$th vertex to the vertex 0; $|M|$ is the cardinality of the set $M$; and the edges $(i,j)$ denote the segments of straight lines between the neighboring points $i$ and $j$ of the grid. As was already noted, problem (1), (2) is NP-hard [3]. In this connection, the present paper proposes an approximate polynomial algorithm of complexity $O(L^2 n)$.

## 2. NECESSARY CONDITIONS FOR EXISTENCE OF SOLUTION

In this section we present two lemmas formulating the obvious necessary conditions for existence of solution.

**Lemma 1.** *For a solution of problem* (1), (2) *to exist, it is necessary that for each terminal there be a path of length $L$ passing only through the free intermediate grid vertices, that is, vertices not occupied by the terminals.*

**Proof of Lemma 1** is self-evident.

**Lemma 2.** *For solution of problem* (1), (2) *to exist, it is necessary that all (rectangular) distances from the terminals to the source be of the same parity (all even or all odd).*

**Proof of Lemma 2.** One can readily see that if the distance from some vertex to the root is even/odd, then any path from this vertex to 0 includes even/odd number of edges. Let, for example, the outermost terminal have even distance $L$. We denote by $l = x_i + y_i$ the odd distance from the $i$th vertex to 0, where $(x_i, y_i)$ are the coordinates of the $i$th vertex. We seek a path of length $L$ from $i$ to 0. Obviously, $\Delta l = L - l > 0$. Therefore, to satisfy constraint (2) for $i$, one needs to add $\Delta l$ edges. Let us assume that we move from $i$ to 0 along a path of length $L$ and that $l_1$ is the number of steps from 0 (upward and rightward). To get to the root, we, obviously, have to make the same number of steps to 0 (downward and leftward). As the result, $\Delta l = 2l_1$; and consequently, the path length has retained its parity, which proves the lemma.

The above conditions are not, of course, sufficient for existence of solution. As can be seen from Fig. 1a, there exists an admissible path of length $L = 7$ from each terminal, but the *admissible tree* does not exist. Indeed, the admissible path from terminal 3 includes a single subpath $\{7,\ 12,\ 11,\ 6, 1,\ 0\}$ (from 3 to 7 one can pass either through 8 or 2). Consequently, the single admissible edge from 5 is (5,10) and then (10,11), but then the path from 5 will be inadmissible because it has length 5. If one moves the terminals 15 and 13, respectively, to the free (intermediate) vertices 17 and 19, then the admissible tree of Fig. 1b becomes possible.

All variants of displacements of terminals make up an enormous set of alternatives. In what follows, we admit displacement at most along one incident edge. If in this case we also assume for definiteness that $L$ is even, then we need to move only the terminals lying at odd distances from the root (Lemma 2). In the concluding section we give a brush treatment to the case where displacement of terminals on two units is admissible.

## 3. METHOD OF SOLUTION

We reduce the original problem to another one for which purpose we construct an $(L + 1)$-level hierarchical structure (HS) with a single vertex 0 at the zero (upper) level and with terminals lying at the $L$th (lowermost) level. The new problem lies in seeking on the HS a tree connecting vertex 0 and those of level $L$ by simple paths (without repetition of vertices) passing only through the intermediate vertices. As the result, all paths will have length $L$.

### *3.1. Probability of Displacement*

We denote by $E$ the set of "even" terminals (having even distances to the root), by $O$ the set of "odd" terminals (having odd distances to the root), and by $I$ the set of intermediate vertices. Before designing the HS, the terminals from the set $O$ must be moved to free neighboring vertices having even distances to the vertex 0. To find the candidates for displacement of the odd terminals,

we calculate for each neighboring intermediate vertex the probability of using it as a new location of an odd terminal. To this end, we first calculate for each terminal $k \in O$ the number $q_k$ of free *admissible* neighbors. The term "admissible" implies here that there exists a simple path through the intermediate vertices to the root. If some $q_k = 0$, then the problem is unsolvable. We assume below that each odd vertex has at least one admissible intermediate vertex. The probability that upon displacement of a neighboring terminal an intermediate vertex $i$ becomes its new location obeys the following obvious formula:

$$\varepsilon_i = 1 - \prod_{k \in V_i} \left(1 - \frac{1}{q_k}\right),$$

where $V_i$ is the set of odd terminals that are neighbors of $i$. If the intermediate vertex $i$ has no neighboring odd terminals, then $\varepsilon_i = 0$. We also assume that $\varepsilon_i = 1$ for all even terminals. Let $I' = \{i \in I | \varepsilon_i > 0\}$ be the set of intermediate vertices having a positive probability of becoming new terminals.

As follows, in particular, from the above formula for $\varepsilon_i$, in the case where $i$ is the single free neighbor vertex of some odd terminal, it necessarily must be the new terminal. We make use of Fig. 2a by way of illustration. Here, vertex 5 has two neighboring odd terminals 6 and 9, that is, $V_5 = \{6, 9\}$, and vertices 6 and 9 have four and two free neighboring vertices, respectively. Therefore, the probability that vertex 5 becomes terminal is $\varepsilon_5 = 1 - (1 - 1/q_6)(1 - 1/q_9) = 1 - (1 - 1/4)(1 - 1/2) = 5/8$. All odd terminals must become intermediate vertices (included into the set $I$), that is, $\varepsilon_i = 0$, $i \in O$. Finally, $I = I \cup O$, $O = \varnothing$, $I' \subseteq I$.

### *3.2. Construction of Hierarchical Structure*

For each even terminal, we assume that there exists an admissible simple path to the source which has length $L$ and passes through the intermediate vertices. We illustrate in Fig. 2b the process of designing the HS for the problem of Fig. 2a. The HS is designed level-by-level beginning from level 0 where only one vertex 0 exists. The next level 1 includes the *intermediate* vertices that

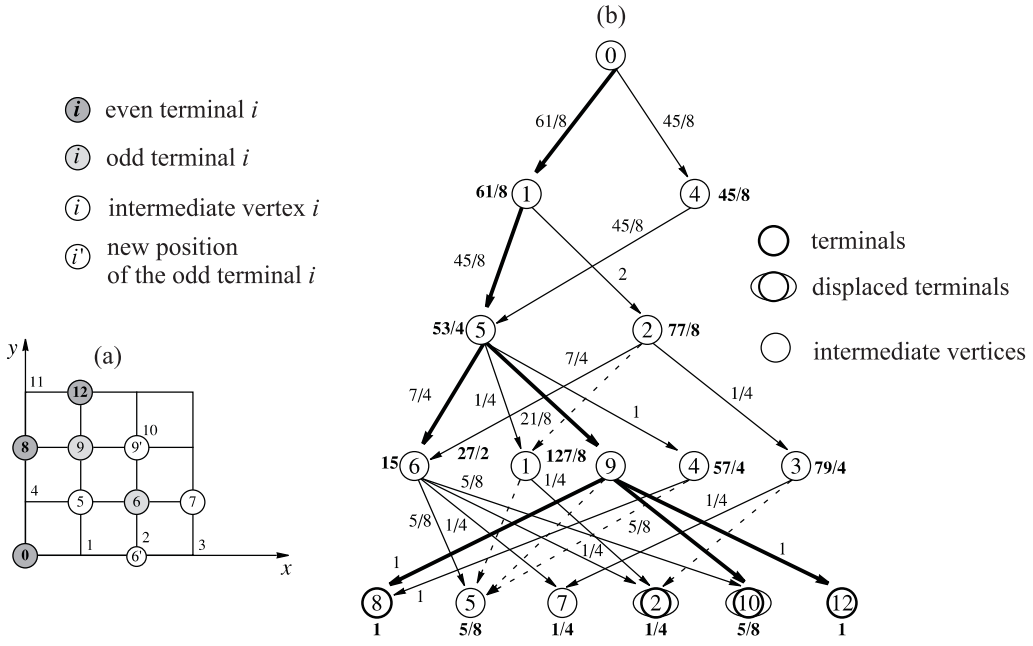


**Fig. 2.** Example of HS design. (a) Initial location of the terminals. (b) Hierarchical structure.

are neighbors of 0 connected to it by edges and making up the set $V_{00}^{+} = m_1$ of successors of the root vertex. For each vertex $i \in m_1$, we denote by $V_{i1}^{-} = \{0\}$ the set of predecessors and store the set of vertices $F_i^1 = \{0\}$ belonging to the path going to it from vertex 0. To check the added edges for admissibility, all the HS vertices will be labelled by the sets $F_i^l$, $l = 1, \ldots, L-1$.

An arbitrary level $l = 2, \ldots, L-1$ consists of the vertices $m_l$. At first, $V_{jl-1}^{+} = \varnothing$, $j \in m_{l-1}$ and $m_l = \varnothing$. Then, for each vertex $i \in I$ we assume that $F_i^l = V$, $V_{il}^{-} = \varnothing$ and consider the neighboring vertices $j$ from $m_{l-1}$. If $i \notin F_j^{l-1}$, then we assume that $V_{il}^{-} = V_{il}^{-} \cup \{j\}$, $V_{jl-1}^{+} = V_{jl-1}^{+} \cup \{i\}$ and $F_i^l = F_i^l \cap F^{l-1}j$.

Finally, the last level $L$ consists of the vertices $i \in V$ for which $\varepsilon_i > 0$. For each vertex $i \in m_L$, we assume that $V_{iL}^{-} = \varnothing$ and check the neighboring vertices $j$ from $m_{L-1}$. If $i \notin F_j^{L-1}$, then we assume that $V_{iL}^{-} = V_{iL}^{-} \cup \{j\}$, $V_{jL-1}^{+} = V_{jL-1}^{+} \cup \{i\}$.

Figure 2b depicts an HS corresponding to the example of Fig. 2a. Here, the odd terminals 6 and 9 can be moved to the vertices 2, 5, 7, or 10. In this example, some edges ((2,1), (1,5), (4,5), (9,5), and (3,2) shown by the dashed line) will be never included in the solution because the corresponding sets $F_j^l$ already involve the vertices under consideration. For example, the vertex $1 \in m_3$ will not be connected with the vertex $2 \in m_2$ because $1 \in F_2^2 = \{0, 1\}$. These edges will be eliminated from the HS.

### *3.3. Algorithm*

The algorithm consists of two stages. At the first stage, weights are assigned to the HS elements. The second stage uses them to solve approximately problem ((1), (2)) on the HS. So, the HS has been constructed. Each vertex $i \in m_L$ of the lower level is assigned the weight $w_i^L = \varepsilon_i$. We calculate the weights of all HS elements. First, we calculate the weights of the vertices of the level $L-1$ and their incident edges connecting the vertices of the $L$th and $(L-1)$st levels. The edge weights are determined first. If an edge enters the vertex $i \in m_L$, then its weight is assumed to be $w_i^L$.

Let us consider an arbitrary $l$th level of the HS. To determine the weight $w_i^l$ of the vertex $i \in m_l$, we sum the weights of the incident edges outgoing to the vertices of level $l+1$, that is, assume that $w_i^l = \sum\limits_{j \in V_{il}^{+}} w_{ij}^{l+1}$. The weight of each edge entering $i$ from above is assumed to be $w_i^l$. As the result, by moving upward one assigns weights to all the HS vertices and edges. In Fig. 2b, weights are shown at the corresponding edges.

The second stage resembles the Dijkstra algorithm [4], but during its execution one must keep in mind not only the label of a vertex, but also the list of the already passed vertices, which is required to prevent cycling in the tree. Initially, the same vertex can be included into different HS levels, but in the desired tree this is forbidden.

We describe the second stage in more detail. At the beginning, the constructed tree is $S = (V, A)$, where $V = \{0\} \cup m_1$ and $A = \{(0, i) | i \in m_1\}$. To each vertex $i \in m_1$, we assign a label vector $(w_i, P_i)$, where $w_i = w_{0i}^1$ and $P_i$ is the set of vertices included in this path from 0 to $i$.

For any vertex $i \in m_l$, $1 < l < L$, we choose the best connection (edge) $(j, i)$, $j \in m_{l-1}$, to the preceding level. This edge is chosen because it maximizes $w_k$, where $k \in m_{l-1}$, $i \notin P_k$, and the edge $(k, i)$ exists in the HS. We label the vertex $i$ by $(w_i, P_i)$, where $w_i = w_j + w_{ji}^l$, $P_i = P_j \cup \{i\}$. In Fig. 2b, the weights of vertices are shown in bold characters next to them.

If at some step one fails to connect a vertex $i \in m_l$, $l < L$ with the preceding level, this vertex can be eliminated from $m_l$. If $l = L$, we arrange the vertices from the set $m_L$ in nonascending order of weights $w_i^L = \varepsilon_i$, $i \in m_L$. Vertices of the same weight are arranged according to the number of incoming edges ($|V_{iL}^{-}|$). The highest-weight vertex with the least—among the vertices of the same

weight—number of incoming paths is the first in this ordered list denoted by $R$. The last vertex in the list $R$ is that with the least positive weight and the greatest number of incoming edges.

We continue search using the list $R$ in order to connect first the vertices that are even terminals or can become such with a high probability. Then, those are chosen that have less variants of connection to the source through an admissible path.

We denote by $C$ the set of vertices of the level $L$ that were found to be connected with the root. The first element of this set is chosen as follows. Let the first vertex from $R$ be $i \in R$, and let it be connected to the vertex $j \in m_{L-1}$ (if there exists the desired edge) and $w_j = \max\limits_{i \notin P_k, k \in m_{L-1}} \{w_k\}$. If there is more than one vertex maximizing this expression, then we choose that having the incoming edge of higher weight. Let us assume that $P_i = P_j$. This is a sequence of $L$ vertices in the path from 0 to $i$. We eliminate the vertex $i$ from $R$ and add it to $C$, that is, $R = R\backslash\{i\}$, $C = C \cup \{i\}$. Additionally, if any other path $P_k$, $k \in R$, has a nonempty intersection with $P_i$ and at least one common vertex $v$ belongs to different levels (has different positions in $P_i$ and $P_k$), then to avoid cycling the path $P_{vk}$ from $v$ to $k$ is eliminated from the constructed tree. If any other next vertex $i \in R\backslash C$ belongs to the set $\bigcup\limits_{k \in C} \{P_k\}$, it is eliminated from $R$, that is, we assume that $R = R\backslash\{i\}$.

This procedure is repeated until emptying $R$. Then one has to determine the new positions of the odd terminals. Initially, the entire list of candidates is included in $m_L$. Some of them may be eliminated after constructing the admissible tree. As a result, we obtain the set $C$ of vertices connected to the HS root through admissible paths. Let $C' = C\backslash E$ be the set of remaining intermediate vertices to which the odd terminals can be moved. For any odd terminal $i \in O$, the set $V_i^C$ of intermediate neighboring vertices from $C'$ is known. Consequently, to determine to where the odd terminals must be moved, it suffices to solve the problem of constructing the maximum matching on the bipartite graph [8]. The parts of this graph are the sets $O$ and $C'$, and an edge $(i, j)$ between the vertices $i \in O$ and $j \in C'$ exists if and only if $j \in V_i^C$. In the case under study where the degree of each vertex in $O$ is at most four, complexity of the algorithm [8] to determine the maximum matching is $O(|O|^{3/2}) \le O((n')^{3/2})$.

For the example of Fig. 2, $\{(9, 10), (6, 2)\}$ is one of the maximum matchings. Therefore, terminals 9 and 6 can be moved, respectively, to vertices 10 and 2.

Thus, on the basis of the determined matching the odd terminals were moved to new places, and the tree constructed in the HS can be used. The admissible paths (bold vectors in Fig. 2b) having $L$ arcs each connect the even and new terminals with the root. This tree is readily representable as a solution of the original problem. For the example of Fig. 2a, the corresponding tree is shown by bold lines.

Below, we describe the pseudocode of the algorithm. We assume that all the HS edges are oriented downward from the root.

**Stage 1.** Finding of the tree edges weights for hierarchical structure.

**Step 0.**

**For** the sets of even, $E$, and odd, $O$, terminals, **do**

Calculate the maximum distance $L$ (in the metric $L_1$) from 0 to each terminal.

**enddo**

**For** all vertices $i \in V$ **do**

Calculate the weights $\varepsilon_i$.

**enddo**

Construct the HS.

**Step 1.** Calculation of arc weights.

**For** each level $l = L-1, \ldots, 1$ **do**

**For** each vertex $i \in m_l$ **do**

Determine the sum of weights of arcs going out of the vertex $i$ and assume that the weight of the arc coming into $i$ is equal to this sum.

**enddo**

**enddo**

**Stage 2.** Construction of the solution.

**For** each level $l = 1, \ldots, L-1$ **do**

**For** each vertex $i \in m_l$ **do**

Determine the best admissible connection of the vertex $i$ to the preceding level.

Calculate the weight of the vertex and store the partially constructed path.

**enddo**

**enddo**

For the ordered set $R$ of vertices the level $L$

**As long as** $R \neq \varnothing$ **do**

For the first vertex $i \in R$, determine the best admissible arc from the level $L-1$.

Renew the admissible paths by comparing with $P_i$ and assume that

$R = R \setminus \{i\}$.

**enddo**

For $O$ and $I'$, determine the maximum matching in order to establish the new vertices to which the odd terminals will be moved.

Represent the solution constructed on the HS over the original grid and calculate the cost of the constructed tree.

## 4. EFFICIENCY OF THE METHOD

We determine complexity of the above approach. To reduce the original problem and construct the HS, it suffices to execute $O(L^2 n)$ operations. Indeed, when constructing an HS and checking the list $F_j^{l-1}$, $j \in V_{il}^-$, for each predecessor, at most four admissible edges of each vertex $i \in m_l$ are checked. Obviously, $|F_j^l| \leq O(L)$ and $|m_l| \leq O(n)$. The number of HS levels is $L$. Consequently, complexity of designing an HS is bounded by $O(L^2 n)$. The second stage of the algorithm is not more complex than the first one. Complexity of the matching construction is at most $O(n^{3/2})$ [8]. Therefore, the total complexity of an approximate solution is $O((L^2+\sqrt{n})n) \leq \min\{O(n^2), O(L^4)\}$.

## 5. SPECIAL CASES

This section focuses on some special cases where the optimal solution can be established in a polynomial time. In the first of them, all terminals lie at distance $L$ from the root vertex. Then, the method of [11] can be used. As the result, the minimum Steiner tree of radius $L$ of the complexity $O(n^2)$ will be constructed.

Another special case that the authors would like to discuss here is as follows. Let $L$ be even, all terminals belong to the set $E$ and be located *far* from each other. The meaning of "far" will be explained below. For the time being, let us imagine that a tree of the shortest distances $D$ from the

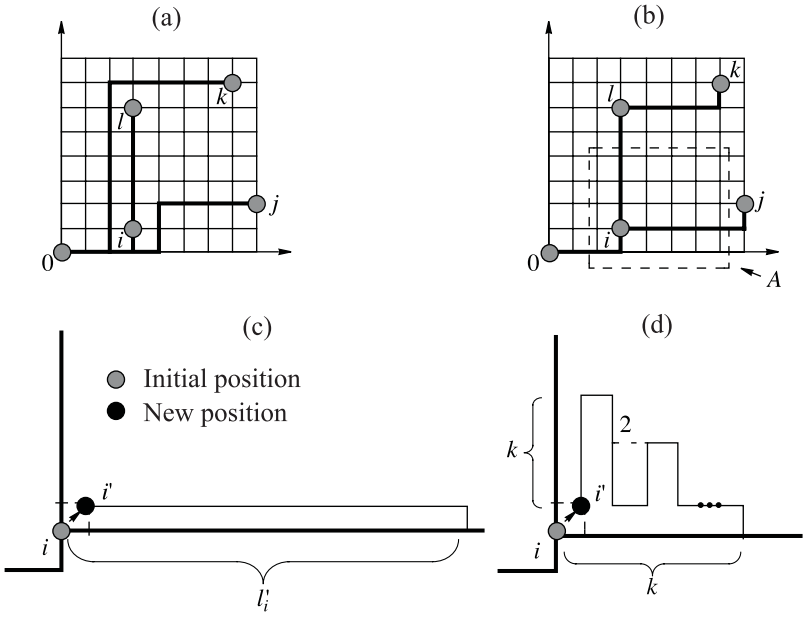


**Fig. 3.** Illustration of increasing the path length in the locality $\delta_i$.

root to all terminals has been constructed and that near each terminal whose distance is smaller than $L$ there is a free (without the elements of the tree $D$) space (locality). Then, inside these localities one can add the missing edges in order to increase the path length to $L$ while retaining $D$. If this is possible, then this procedure can construct an admissible solution.

Let us assume that each terminal $i$ with distance $l_i < L$ has on top right (at "north-east") a free $\delta_i$-locality, that is, that there are neither vertices nor edges of the tree $D$ at the distance $\delta_i$ from the terminal in the direction strictly upward-rightward (excluding only upward or only rightward). If this $\delta_i$-locality allows, then the missing length can be obtained by adding the necessary amount of edges ($2l'_i = \Delta l_i = L - l_i$) in the following manner (Fig. 3). We displace the terminal $i$ one step up and one step right. The new path from the new position of the terminal (denoted by $i'$) will go, for example, $l'_i - 1$ units (steps) to the right, then one step down, and $l'_i$ steps to the left (back) to $i$ (Fig. 3c). As the result, the path from $i'$ to $i$ consists of $l'_i - 1 + 1 + l'_i = 2l'_i$ units, and the entire path from $i'$ to 0 has the length $L$.

Obviously, to realize the path-extension procedure, it suffices to satisfy the inequality $\delta_i \geq l'_i + 1$. Understandably, the $\delta_i$-locality can be used more thriftily. This economical approach is depicted in Fig. 3d where an attempt is made to expand the domain used for extending the path length.

It remains to discuss how to construct a "suitable" tree $D$ of the shortest paths. If the Dijkstra algorithm [4] is used, then the constructed tree can have, for example, the form shown in Fig. 3a. Indeed, all terminals are connected to the root by shortest paths, but do not have free space for local extension of the appropriate paths. To avoid this undesirable situation, one can apply the algorithm of [5, 6] which has the same complexity as that of Dijkstra, but when choosing the current edge, takes into account not only the path length, but also the distance to the vertices of the partial tree and constructs a "suitable" tree $D$. For sufficiently great values of the parameter, this algorithm constructs the tree of shortest paths of the minimum total weight (Theorem 1 [5]). Moreover, the situation of Fig. 3a will not occur because the nearest to the root vertices are included in the partial tree before the more distant ones (Lemma 1 [6]). For example, the vertex $j$ in Fig. 3a will be connected to the vertex $i$, and the vertex $l$, to the vertex $k$. There is only one more requirement that should be added to the rules of the algorithm. Any pair of vertices can be connected by a path consisting at most of two segments of straight lines. Moreover, if upon "moving" along the path

from the root, an edge consists of two segments, then the first segment must be horizontal, and the second, vertical (lower angle of turn). For example, the edge $(k, l)$ in Fig. 3b is admissible, whereas $(0, k)$ in Fig. 3a is an inadmissible edge. The above results give rise to the following theorem.

**Theorem.** *If for each terminal $i$ there exists a free rectangular locality having sides at least $l'_i + 1$ and located upward-rightward, then the admissible solution of problem* (1), (2) *exists and can be constructed with complexity $O(n^2)$.*

## 6. EXPERIMENTAL RESULTS

The proposed method was validated by a numerical experiment for small size ($n = 40$, $n' \in [6, 15]$, $L \in [3, 7]$) where the root and terminals are located arbitrarily over the rectangle. Examples having no admissible solution were not considered. The experiment was aimed at (i) estimating the ratio error, (ii) determining the number of cases where the proposed approach failed to construct an admissible solution, and (iii) comparing the proposed algorithm with the existing counterparts such as the algorithm based on successive construction of matchings.

Prior to presenting the results of experiment, we recall operation of the matching-based algorithms [1, 2, 9]. They construct successively the maximum minimum-weight matchings. "Maximality" means that the maximum number of vertex pairs is connected by nonadjacent edges; and "minimum weight" means that among all maximum matchings the sum of weights included in the desired edge matching is minimal.

This matching is first constructed for the terminals. At the second step, the matching is constructed for the middles of edges constructed at the preceding step. Then, on each new edge one seeks a point equidistant from the terminals, and the matching is constructed now for these points, and so on until a connected tree graph is obtained. On the last constructed edge, one needs to find a point equidistant from all terminal, which will the tree root.

It is well known that there are rather many shortest paths between two points over the rectangular grid. With the matching-based approach, it is not clear which path must be used as the representation of each edge. Yet, the choice of path is a sufficiently important issue because of its impact on the subsequent edge routing. It may turn out that an unsuccessful choice of some paths at the earlier steps would make the admissible tree infeasible. Another disadvantage of these algorithms lies in that the root of the desired tree is determined only by the algorithm and is unknown in advance.

In the numerical experiment, the root vertex was placed in advance (at different points of the rectangle). Therefore, in the case where the method of successive construction of matchings resulted in another position of the root, we added to the weight of tree the length of the admissible shortest path between the fixed and obtained positions of the source.

Much time was spent on determining a counterexample for which the method proposed in this paper fails to determine an existing admissible solution (for the aforementioned size), that is, if a solution existed, then the proposed algorithm constructs the admissible solution in 100% of cases of which 80% were optimal. The mean mismatch between "our" and the optimal solutions was at most 1.5%.

The matching-based approach failed to find the admissible solution in 35% of cases. In 15% of cases, it constructed better (smaller weight of the tree) solutions, in 35%, solutions of the same weight, and in 50%, solutions of greater weight. Thus, the approach proposed here in 85% of cases was not worse than the matching-based one. The results of the numerical experiment for twenty randomly chosen examples are compiled in the table where $W_A$ is the weight of tree constructed as proposed in this paper; $W_M$ is the weight of tree constructed by the matching-based algorithm (the asterisk $*$ in column means that $L$ was increased up to seven); "fail" in the column $W_M$ means that

Comparison of the algorithms on 20 examples ($n = 40$)

| Example # | Number of terminals | $L$ | $W_A$ | $W_M$ | $W^*$ | $\frac{W_A - W^*}{W^*} \times 100\%$ |
|---|---|---|---|---|---|---|
| 1 | 7 | 4 | 18 | 17 | 17 | 5.88 |
| 2 | 13 | 4 | 30 | 27 | 27 | 11.1 |
| 3 | 4 | 4 | 8 | 8 | 8 | 0 |
| 4 | 3 | 5 | 11 | 11* | 11 | 0 |
| 5 | 7 | 4 | 16 | 16 | 16 | 0 |
| 6 | 7 | 4 | 14 | 14 | 14 | 0 |
| 7 | 7 | 3 | 14 | 13 | 13 | 7.7 |
| 8 | 8 | 3 | 14 | 14 | 14 | 0 |
| 9 | 8 | 4 | 16 | fail | 16 | 0 |
| 10 | 8 | 4 | 17 | 19 | 17 | 0 |
| 11 | 8 | 6 | 27 | fail | 27 | 0 |
| 12 | 8 | 7 | 22 | fail | 22 | 0 |
| 13 | 8 | 7 | 20 | fail | 20 | 0 |
| 14 | 8 | 5 | 18 | 20 | 18 | 0 |
| 15 | 8 | 3 | 14 | 14 | 14 | 0 |
| 16 | 8 | 4 | 17 | 18 | 17 | 0 |
| 17 | 10 | 5 | 22 | fail | 22 | 0 |
| 18 | 7 | 5 | 19 | fail | 18 | 5.5 |
| 19 | 9 | 6 | 20 | 20 | 20 | 0 |
| 20 | 9 | 5 | 22 | fail | 22 | 0 |

the matching-based algorithm failed to construct an admissible solution despite its existence; and $W^*$ is the weight of the optimal tree. An additional advantage of our approach over that based on matchings lies in the possibility of displacing terminals if the problem has no admissible solution.

## 7. CONCLUSIONS AND SOME REMARKS

The above approach, of course, is just an expedient heuristics enabling one to construct approximate solutions. In the event of unsuccessful outcome, one can fail to obtain even an admissible tree notwithstanding the fact that it exists.

The idea of choosing in the course of operation of the algorithm the maximum-weight edges allows one to reduce the weight of the constructed tree because it is an attempt to use the same edge in different paths from the source to the terminals. Sometimes, however, this may lead to a situation where none of the admissible trees can be constructed. To avoid this, one can add in the course of the second step any randomly chosen admissible edge, rather than seek a maximum-weight path in the HS.

Another application of the proposed method is the method of implicit enumeration (for example, in the branch-and-bound method [10]). Branching of the HS can be easily organized. At each level one can choose the edge to be or not included in the constructed tree. For the partial tree, one can use the proposed algorithm to establish a new record. As the result, some branches can be truncated.

It would be, of course, desirable to establish some *a priori* estimates of accuracy rating to check quality of the approximate solution. Usually, for the majority of problems involved in applied problems this is impossible. Therefore, the *a posteriori* analysis seems to offer the only way to estimating approaches to such problems. Here, the branch-and-bound method can be used to obtain the optimal solution and calculate its ratio error. Studies of the proposed algorithm are

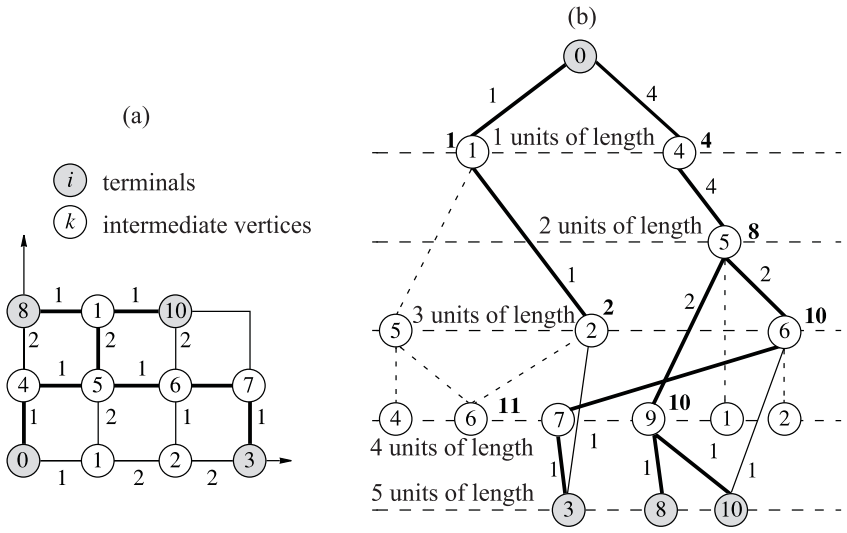


**Fig. 4.** Example with arbitrary lengths of edges. (a) Position of the terminals. ($L = 5$. Lengths of the edges are shown near them.) (b) HS and the optimal tree (bold lines). Weights are shown near the edges and vertices.

going on, and we plan to carry out a more profound *a posteriori* analysis. Anyway, the above numerical analysis suggests that the algorithm is quite satisfactory.

Another obvious generalization of this approach lies in using it when the terminals lie in arbitrary places of the plane and not only in the first quadrant. The proposed method is applicable to this general case as well. Indeed, we never made any use of the information about the domain of location of the terminals. The space where there may be the elements of the desired tree is easily bounded by $L$. Only the grid vertices lying within the circle of radius $L$ and centered at 0 get numbers and can be included in the desired tree.

If the proposed method fails to determine an admissible solution, the terminals can be displaced to other points of the grid lying at distances not exceeding two. For example, if $L$ is even, then the even terminals can be moved by two units, and the odd terminals, by one unit. We propose to divide the process of displacement into two stages. First, it is recommended to solve the problem where only the odd terminal are displaced. If an admissible solution is established, then stop. Otherwise, denote by $T'$ the partial tree constructed at the previous stage. The even terminals that remain disconnected make up the set $U$ of candidates for displacement. They can be moved successively to the *free* places within the simplex bounded by the straight line $x + y = L$. Here, the term "free" means that these places contain neither terminals nor intermediate vertices included in the constructed subtree $T'$. The intermediate vertices having the minimum of neighbors (at distance 2) from the even terminals are the preferable candidates for being terminals. Then, one may try again to solve the problem using the proposed algorithm. This procedure can be reiterated until an admissible solution is obtained.

Another technique that can be used to displace the terminals from the set $U$ is the metaheuristics simulating behavior of ants. Let us imagine an ant beginning its "voyage" from an arbitrary disconnected terminal belonging to $U$. It can move without turns in any direction until it passes two edges of the grid or meets a *barrier* such as terminal or intermediate point included in $T'$. If the barrier blocks further rectilinear motion, then in the case where the ant passed two edges it moves one edge back, otherwise it turns in any direction in which it did not move before. If all paths from the current point where the ant stays were checked, then it moves back along the edge that led it to this point. If the ant is at a free point at the distance of two edges from the initial position of the disconnected terminal, then this vertex will be used for displacing the terminal. Otherwise, if

the ant checked all admissible paths of length two and found no free place, the terminal remains in the same place.

The ants "staying" at various disconnected terminals can "travel" successively one after another starting from all terminals of the set $U$ and passing one edge at a cycle. Since the grid graph has the degree four, the maximum number of steps (checked edges) is bounded by 16 for each terminal. Therefore, the number of trials (steps) of the ants is bounded by $O(|U|) \le O(n')$. Consequently, this procedure does not increase complexity of the above algorithm.

We conclude by noting that the proposed approach can be extended to the case where each grid edge has a non-unit, but an arbitrary *integer* length. With this generalization, one can also determine the *minimum* distance $L$ to the outermost terminal. The corresponding HS will have $(L + 1)$ levels. An arbitrary vertex will belong to the level $l$ of the HS if there is an admissible (passing though the intermediate vertices) path of length $l$ that goes from the root to it. The HS can be constructed in a way similar to the aforementioned method. Yet, in the HS the edge can connect vertices not only of the neighboring levels. Depending on the length of the corresponding grid edge, an HS edge can connect vertices of different levels. At the second stage of the algorithm, the least-weight edge is chosen if there is more than one way of connection. Figure 4 depicts an example of constructing an HS for the last generalization. The bold lines show the tree constructed by the algorithm. It is admissible and even optimal.

## REFERENCES

1. Boese, K.D. and Kahng, A.B., Zero-skew Clock Routing with Minimum Wire-length , *Proc. 1st Annual IEEE Int. ASIC Conf. and Exhibit*., 1992, pp. 17–21.
2. Cong, J. and Koh, C.K., Minimum-cost Bounded Skew Clock Routing, *Int. Symp. Circuits and Systems*, 1995, pp. 215–218.
3. Kong, J., Kahng, A., and Robins, G., Matching Based Methods for High-Performance Clock Routing, *IEEE Trans. CAD-IC*, 1993, vol. 12(8), pp. 1157–1169.
4. Dijkstra, E.W., A Note on Two Problems in Connection with Graphs, *Numer. Math.*, 1959, no. 1, pp. 269–271.
5. Erzin, A.I. and Kim, S.H., Polynomial Algorithm for Min-cost Delay-constrained Multicast Routing Problem in Networks, *Proc. 1st Int. Conf. Inf. Commun. and Signal Proc. (ICICS'97)*, Singapore, 1997, pp. 1805–1809.
6. Erzin, A.I. and Cho, J.D., A Deep-submicron Steiner Tree, *Math. Comput. Modelling*, 2000, no. 31, pp. 215–226.
7. Garey, M.L. and Johnson, D.S., *Computers and Intractability: A Guide to the Theory of NP-Completness*, San Francisco: Freeman, 1979. Translated under the title *Vychislitel'nye machiny i trudnoreshaemye zadachi*, Moscow: Mir, 1982.
8. Hopcroft, J.E. and Karp, R.M., A $n^{5/2}$ Algorithm for Maximum Matching Bipartite Graphs, *J. SIAM Comp.*, 1973, no. 2, pp. 225–231.
9. Kahng, A., Cong, J., and Robins G., High-performance Clock Routing Based on Recursive Geometric Matching, *Proc. Design Autom. Conf., IEEE/ACM*, 1991, pp. 322–327.
10. Papadimitriou, C.H., *Computational Complexity*, New York: Addison-Wesley, 1994.
11. Tellez, G. and Sarrafzadeh, M., On Rectilinear Distance-Preserving Trees, *Proc. IEEE Int. Symp. Circuits and Systems (ISCAS'95)*, Washington, 1995, vol. 1, pp. 163–166.

*This paper was recommended for publication by V.M. Vishnevskii, a member of the Editorial Board*